mtool estimators floating precision

Simulation Settings

The cause-specific hazards of the outcome of interest and the competing risk follow proportional hazards models, specifically:

$$\alpha_{01} = 0.5t\ \exp(\beta_{01} Z)$$

$$\alpha_{02} = t\ \exp(\beta_{02} Z)$$

where both cause-specific hazards have the form of a Weibull distribution and a common set of covariates. Cause 1 is the one of interest with an incidence rate of 54 % while cause 2 has an incidence rate of 28 % with a common uniform censoring rate of $\sim 15\%$.

```
knitr::opts_chunk$set(cache = T)

# independent normal variables
p <- 20
n <- 400
rho <- 0.5

# Very sparse case
# Common betas for both competing risks
beta <- c(0.5, rep(0, 18), 0.5)
zero_ind1 <- which(beta == 0)
nonzero_ind1 <- which(beta != 0)

# Generate iid X's
X <- matrix(rnorm(p*n), nrow = n, ncol = p)
# XB matrix
suma <- X %*% beta
```

```r
# Function to generate survival times
create.times <- function(n, ch, sup.int = 100) {
  times <- numeric(n)
  i <- 1
  while (i <= n)
  { u <- runif(1)
  if (ch(0, -log(u)) * ch(sup.int, -log(u)) < 0)
  { times[i] <- uniroot(ch, c(0, sup.int), tol = 0.0001, y= -log(u))$root
  i <- i + 1
  }
  else {
    cat("pos")
  }}
  times
}

# binomial probability of cause 1
binom.status <- function(ftime, n, a01, a02, size = 1)
{ prob <- a01(ftime) / (a01(ftime) + a02(ftime))
out <- rbinom(n, size, prob)
out }


# Cause-specific proportional hazards
times <- vector()
f.status <- vector()
for (i in seq_len(n)) {
alpha.1 <- function(t) { ((0.5*t)*exp(suma[i])) }
alpha.2 <- function(t) { t*exp(suma[i]) }

cum.haz <- function(t, y) { stats::integrate(alpha.1, lower=0.001, upper=t,
                                             subdivisions=1000)$value +
 stats::integrate(alpha.2, lower=0.001, upper=t,
                 subdivisions=1000)$value - y }
times[i] <- create.times(1, cum.haz)
f.status[i]<- binom.status(times, 1, alpha.1, alpha.2) + 1
}

# Censoring
cens.times <- runif(n, 0, 6)
```

```r
# Censoring in status variable
f.status <- as.numeric(times <= cens.times) * f.status

prop.table(table(f.status))
```

```
f.status
     0      1      2 
0.1775 0.5450 0.2775 
```

```r
# times with censoring
times <- pmin(times, cens.times)

# Dataset
sim.dat <- data.frame(time = times, status = f.status)


sim.dat <- cbind(sim.dat, X)

colnames(sim.dat)[3:22] <- paste0("X", seq_len(p))

head(sim.dat)
```

```
       time status         X1          X2          X3          X4         X5
1 0.9573900      0 -0.2833227  0.49315714 -0.39741951  2.03625495 -0.7032940
2 1.1799018      2  0.3232338  1.71616792 -0.46033546  0.02252274  0.3078185
3 0.8318170      1  1.7249138 -0.81680140 -0.56385883  0.94053695 -0.2924841
4 0.7128943      1  0.2462367 -0.43702486 -1.07616818 -0.11868349 -0.6220214
5 0.3465138      0 -1.2528001 -0.07444809  0.02754773  1.39543413  1.3102259
6 1.8083344      1 -0.3435119  2.05859431 -0.81325204  0.95979719  0.8135085
          X6         X7         X8         X9        X10        X11        X12
1  1.3812699 -1.4583283  1.5471274 -0.4759714 -1.1536913  0.9986666 -0.3782998
2  1.0390017  0.6989621 -1.1535188  1.1049859 -0.2081242  1.4297879 -0.9685742
3 -0.5955198  0.5454273 -1.0567628 -0.9781166 -2.9286698  0.8496133  0.7936185
4  0.8488174 -1.3804222  0.6297198 -1.8816258  0.3714670  0.6091019  0.3082478
5 -2.9057474 -0.2645360 -0.1934075 -0.2205454 -1.4219697  1.3002518 -0.5440021
6 -1.1784667  0.9474378  1.9692580  0.1082714 -1.4709294 -0.3398231  1.6439283
         X13        X14          X15       X16        X17         X18
1 -0.5719765  0.7106656 -0.805159653 0.1649621 -0.7901370 -0.69944603
2 -0.2328593  0.2757856 -1.059329592 1.6586922  0.1182002 -1.81396615
3  0.4405046 -0.7697308 -0.007615668 0.2647511  0.2657026 -1.42275830
```

```
4 -0.0140577  0.6820971  0.330629847 0.1664212 -1.5886231  0.01914981
5  1.3361019  1.3378298 -1.418223609 0.1806255  0.4111172  0.64280065
6 -0.7253748 -0.8876671 -1.779505778 1.7888353  0.1064518 -1.25743280
         X19         X20
1  0.3503175 -0.03108879
2 -1.2123014  0.63416536
3  2.5450875 -0.80637090
4 -0.2476327  0.01410537
5 -2.5503276  1.55633581
6  1.4039245 -0.41585924
```

2. Floating precision of mtool estimators

```r
# Split into training and test sets
train.index <- caret::createDataPartition(sim.dat$status, p = 0.70, list = FALSE)
train <- sim.dat[train.index,]
validation <- sim.dat[-train.index,]

surv_obj_train <- with(train, Surv(time, as.numeric(status), type = "mstate"))

cov_train <- cbind(train[3:22])

# Create case-base dataset
cb_data_train <- create_cbDataset(surv_obj_train, as.matrix(cov_train))

# Apply to validation set
# First fit
lambdagrid <- rep(0.01, 150)
cvs_res <- mclapply(lambdagrid, function(lambda_val) {
fit_val1 <- fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                                        lambda = lambda_val , alpha = 1)
 }, mc.cores = 4)

lambdagrid <- rep(0.009, 150)
cvs_res1 <- mclapply(lambdagrid, function(lambda_val) {
fit_val1 <- fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                                        lambda = lambda_val , alpha = 1)
 }, mc.cores = 4)
```

```r
non_zero_coefs <-  unlist(mclapply(cvs_res,
function(x) {return(x$no_non_zero)}, mc.cores = 4))

non_zero_coefs1 <-  unlist(mclapply(cvs_res1,
function(x) {return(x$no_non_zero)}, mc.cores = 4))

# Display different outputs
cvs_res[[1]]
```

```
$coefficients
22 x 2 sparse Matrix of class "dgCMatrix"

 [1,]  0.1188583  0.03100094
 [2,]  .          .
 [3,]  .          .
 [4,]  .          .
 [5,]  .          .
 [6,]  .          .
 [7,]  .          .
 [8,]  .          .
 [9,]  .          .
[10,]  .          .
[11,]  .          .
[12,]  .          .
[13,]  .          .
[14,]  .          .
[15,]  .          .
[16,]  .          .
[17,]  .          .
[18,]  .          .
[19,]  .          .
[20,]  0.1460524  0.10126200
[21,]  0.3274119  0.39797117
[22,] -0.1295394 -0.41744877

$no_non_zero
[1] 8
```

```r
cvs_res[[which(non_zero_coefs != 8)[1]]]
```

```
$coefficients
22 x 2 sparse Matrix of class "dgCMatrix"

 [1,]  1.188379e-01  0.03094678
 [2,]  .             .
 [3,]  .             .
 [4,]  .             .
 [5,]  .             .
 [6,]  .             .
 [7,]  .             .
 [8,]  .             .
 [9,]  .             .
[10,]  .             .
[11,]  .             .
[12,]  .             .
[13,]  .             .
[14,]  .             .
[15,]  4.438434e-08  .
[16,]  .             .
[17,]  .             .
[18,]  1.443084e-08  .
[19,]  .             .
[20,]  1.461072e-01  0.10135319
[21,]  3.275139e-01  0.39809244
[22,] -1.295302e-01 -0.41744088

$no_non_zero
[1] 10
```

```r
prop.table(table(non_zero_coefs))
```

```
non_zero_coefs
          8           9          10 
0.946666667 0.046666667 0.006666667 
```

```r
prop.table(table(non_zero_coefs1))
```

```
non_zero_coefs1
          8           9          10          11 
0.860000000 0.106666667 0.026666667 0.006666667 
```

Fixing the issue

Increased the `tolerance` from 1e-4 to 1e-5 and the `niter_inner_mtplyr` to 15 (from 5) (number of stochastic updates used to estimate the full gradient)

```
# Penalized Multinomial Logistic Regression
mtool.MNlogistic <- function(X, Y, offset, N_covariates,
                             regularization = 'l1', transpose = F,
                             lambda1, lambda2 = 0, lambda3 = 0,
                             learning_rate = 1e-4, tolerance = 1e-5,
                             niter_inner_mtplyr = 15, maxit = 100, ncores = -1,
```

Running the same test again with new function

```
lambdagrid <- rep(0.02, 150)
cvs_res <- mclapply(lambdagrid, function(lambda_val) {
fit_val1 <- fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                                          lambda = lambda_val , alpha = 1)
 }, mc.cores = 4)

non_zero_coefs <-  unlist(mclapply(cvs_res,
function(x) {return(x$no_non_zero)}, mc.cores = 4))

prop.table(table(non_zero_coefs))
```

```
non_zero_coefs
6
1
```

```
lambdagrid <- rep(0.009, 150)
cvs_res1 <- mclapply(lambdagrid, function(lambda_val) {
fit_val1 <- fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                                          lambda = lambda_val , alpha = 1)
 }, mc.cores = 4)

non_zero_coefs1 <-  unlist(mclapply(cvs_res1,
function(x) {return(x$no_non_zero)}, mc.cores = 4))

prop.table(table(non_zero_coefs1))
```

```
non_zero_coefs1
7
1
```

```r
#' Multinomial deviance
#'
#' @param cb_data Output of \code{create_cbDataset}
#' @param fit_object Output of \code{fit_cbmodel}
#' @return Multinomial deviance
multi_deviance <- function(cb_data, fit_object) {
  X <- as.matrix(cbind(cb_data$time, cb_data$covariates))
  fitted_vals <- as.matrix(X %*% fit_object$coefficients)
  pred_mat <- VGAM::multilogitlink(fitted_vals,
                                   inverse = TRUE)
  # Turn event_ind into Y_mat
  Y_fct <- factor(cb_data$event_ind)
  Y_levels <- levels(Y_fct)
  Y_mat <- matrix(NA_integer_, ncol = length(Y_levels),
                  nrow = nrow(X))
  for (i in seq_len(length(Y_levels))) {
    Y_mat[,i] <- (Y_fct == Y_levels[i])
  }

  dev <- VGAM::multinomial()@deviance(pred_mat, Y_mat,
                                      w = rep(1, nrow(X)))

  return(dev)
}
```